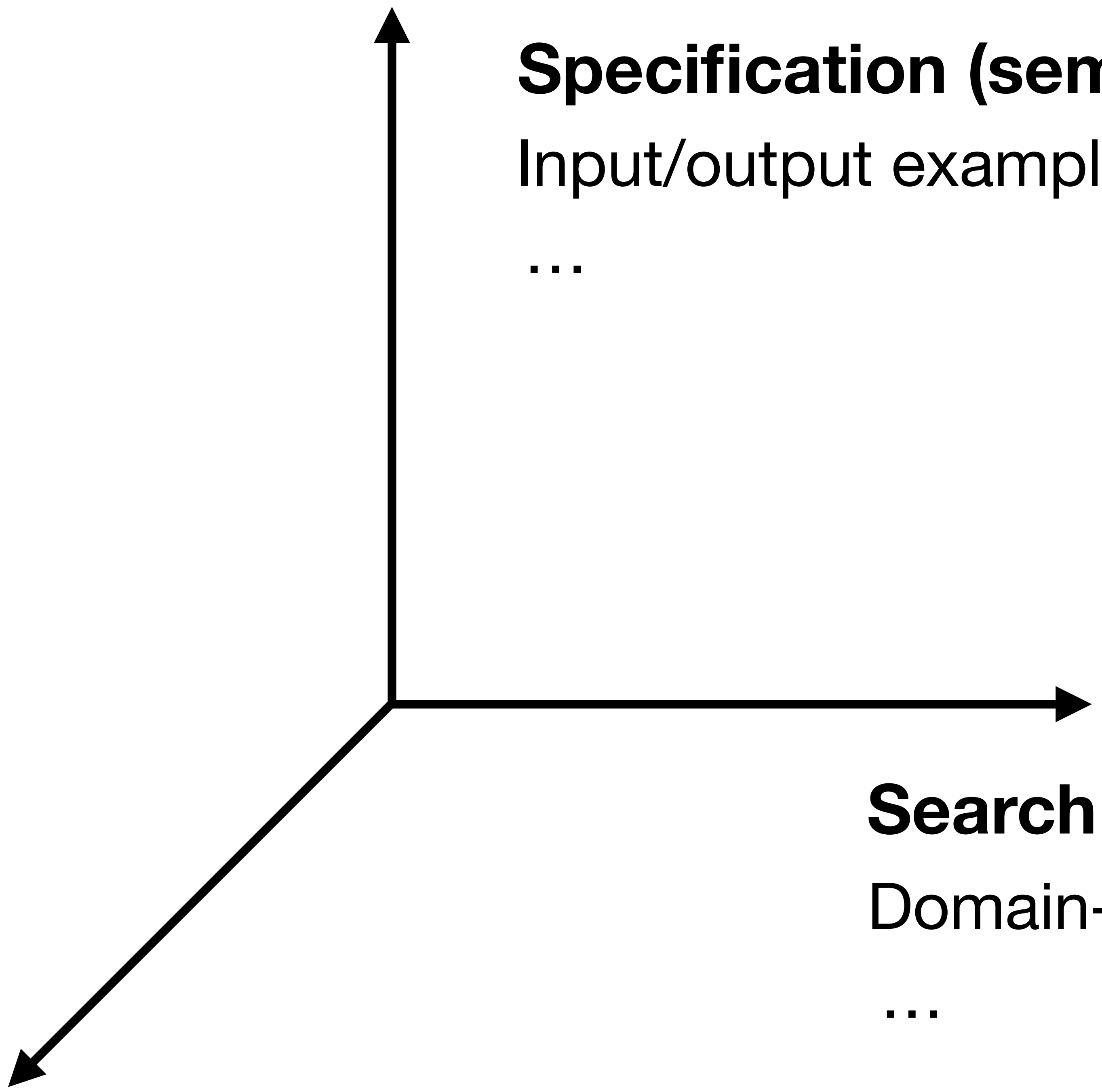
Program Reasoning

13. Representation-based Search

Kihong Heo

KAIST

Dimensions in Program Synthesis



Specification (semantic constraint)

Input/output examples

…

Search strategy

Enumerative search

Representation-based search

…

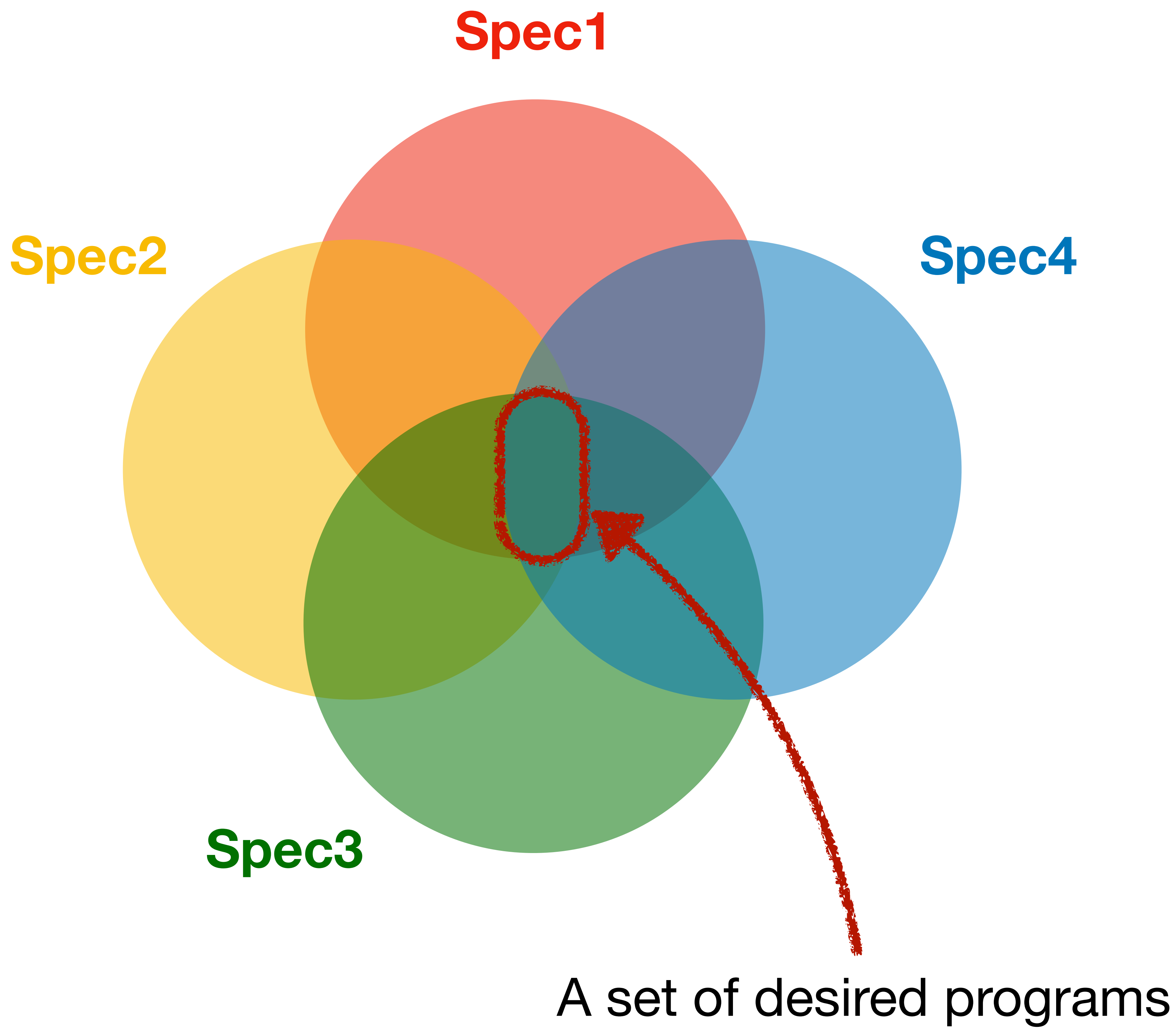
Search space (syntactic constraint)

Domain-specific language

…

Goal: Finding a Set of Programs

- So far: search for a single solution
  - Enumerate one-by-one
- This lecture: search for a set of solutions
  - Return multiple results then rank them
  - Space-efficient search



Representation-based Search

- Idea:
  - Build a data structure that concisely represents a set of programs
  - Extract solutions from that data structure
- Two well-known methods
  - Version space algebra (VSA)
  - Finite tree automata (FTA)

Version Space

- Hypothesis: a function that takes an input and an output
- Hypothesis space *H*: a set of all hypotheses (i.e. programs)
- Version space $VS_{H,D} \subseteq H$: a set of programs that satisfy the examples in the given dataset
    - $D = \{(in_i, out_i)\}_i$: a set of input-output examples
    - $h \in VS_{H,D} \iff \forall i, o \in D \,.\, h(i) = o$

Version Space Algebra

- A set of operations to manipulate and compose version space
- Operations on version spaces:
  - learn(i, o): construct a version space of functions consistent with (i, o)
  - $VS_1 \cap VS_2$, $VS_1 \cup VS_2$: intersection and union of two version spaces
  - pick VS: pick a function from version space VS
- Synthesis idea: use of compact symbolic representation for the version spaces

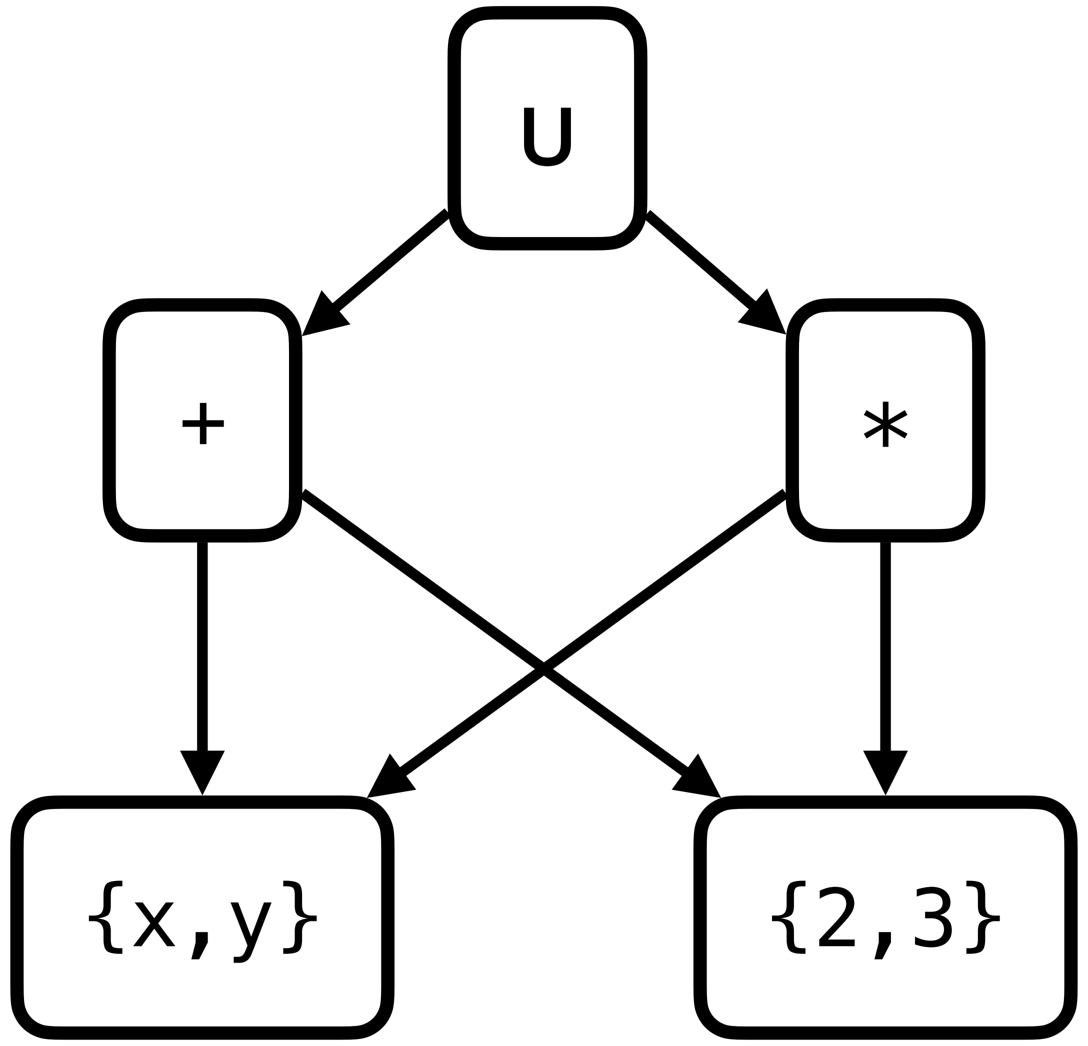
Syntax of VSA

- Grammar of VSA

$$\widetilde{P} ::= \{P_1, \ldots, P_k\} \mid \mathbf{U}(\widetilde{P}_1, \ldots, \widetilde{P}_k) \mid F_{\bowtie}(\widetilde{P}_1, \ldots, \widetilde{P}_k)$$

- Example: {`x+2, x+3, y+2, y+3, x*2, x*3, y*2, y*3`}

$\mathtt{\mathbf{U}(+_{\bowtie}(\{x,y\},\ \{2,3\}),\ *_{\bowtie}(\{x,y\},\ \{2,3\}))}$

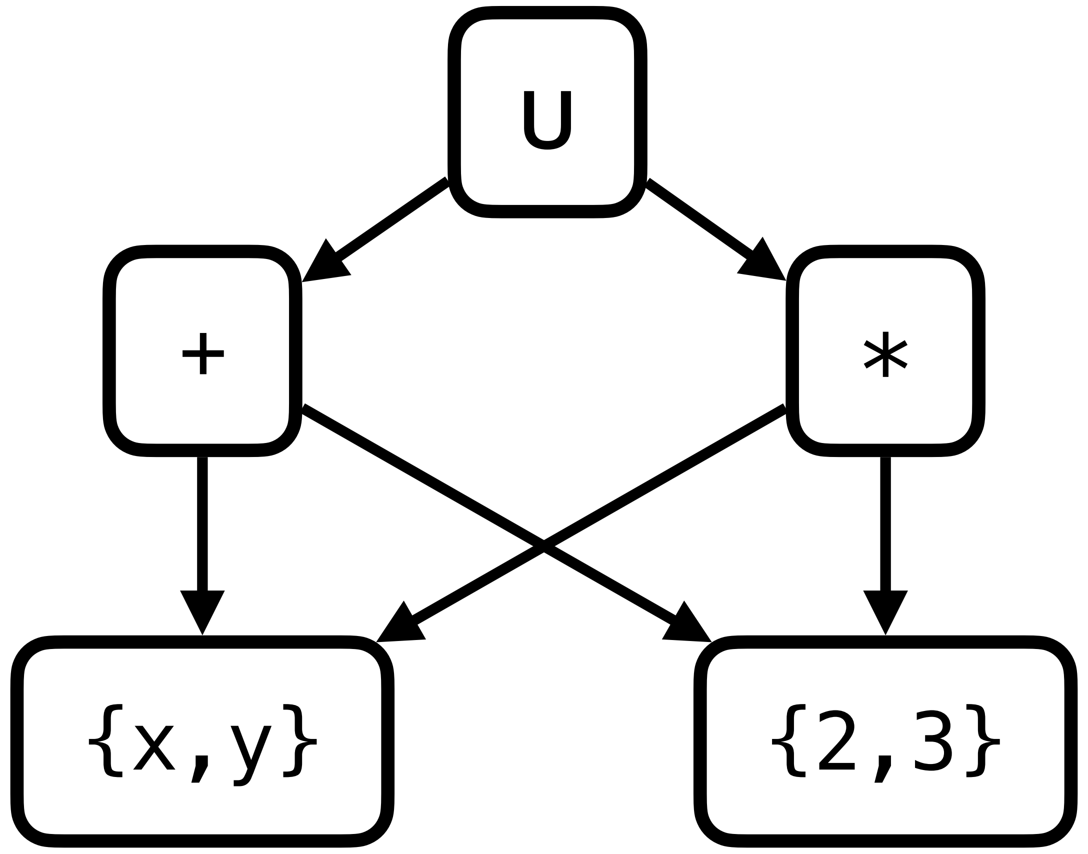


Semantics of VSA

- A program P is an element of a VSA

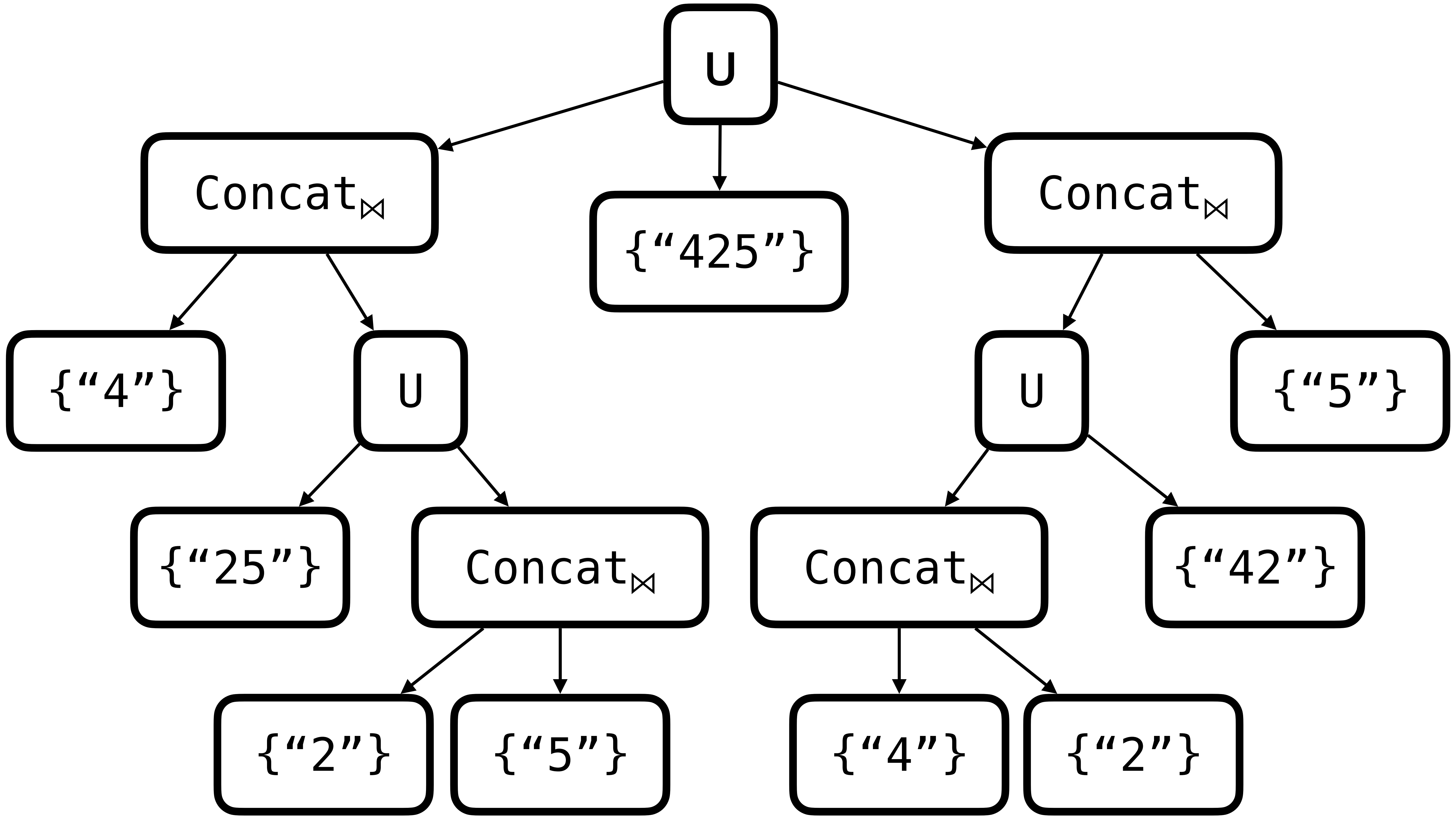
$$
\begin{array}{ll}
P \in \{P_1, \ldots, P_k\} & \exists j. P = P_j \\
P \in \mathbf{U}(\widetilde{P}_1, \ldots, \widetilde{P}_k) & \exists j. P \in \widetilde{P}_j \\
P \in F_{\bowtie}(\widetilde{P}_1, \ldots, \widetilde{P}_k) & P = F(P_1, \ldots, P_k) \wedge \forall j. P_j \in \widetilde{P}_j
\end{array}
$$

- Example:



Example

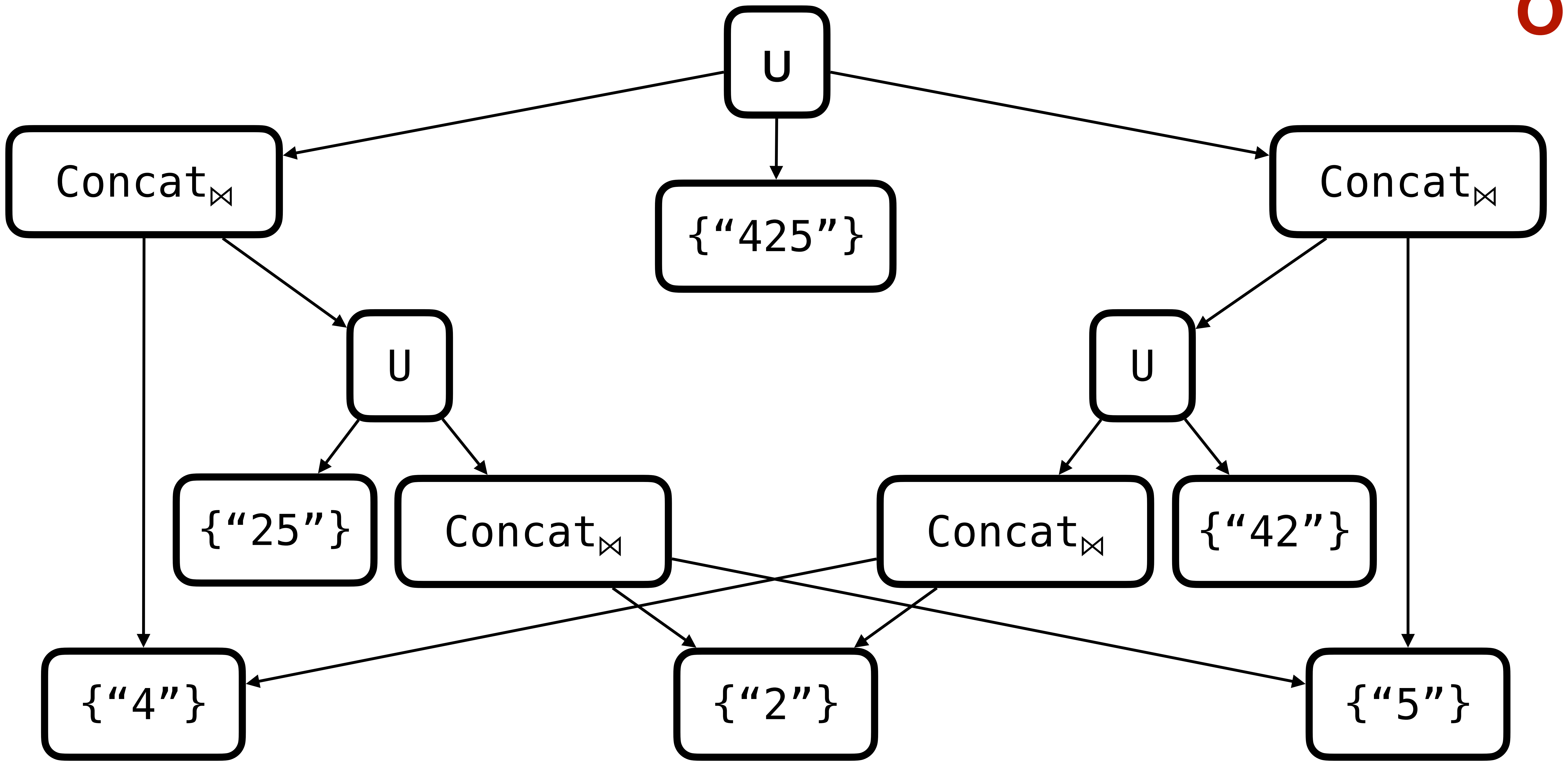
- Grammar $S \rightarrow ConstStr \mid \text{Concat}(S, S)$
- A set of program that returns "425"



Example

- Grammar $S \rightarrow ConstStr \mid \text{Concat}(S, S)$
- A set of program that returns “425”

Optimization!

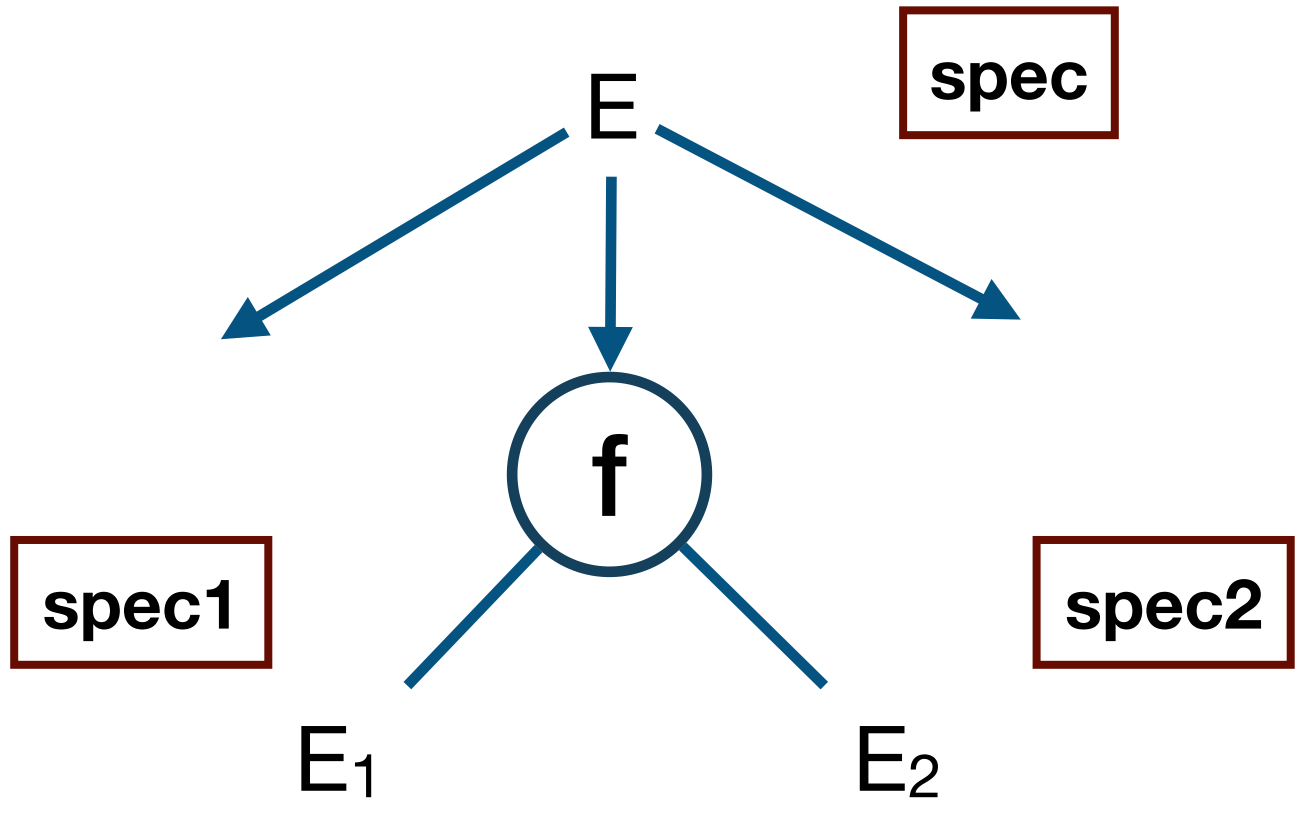


Efficiency

- Represent potentially exponential program sets in polynomial space
  - $V(VSA)$: # nodes in VSA
  - $|VSA|$: # programs in VSA
  - $V(VSA) = O(\log|VSA|)$
- E.g., millions of programs => hundreds of nodes

TDP with VSA

- Given a spec and a production, infer specs for subprograms (divide-and-conquer)
  - When `f<E1, E2, …, En>` (In)= Out where E_i is a subprogram
  - What is the spec for each E_i?

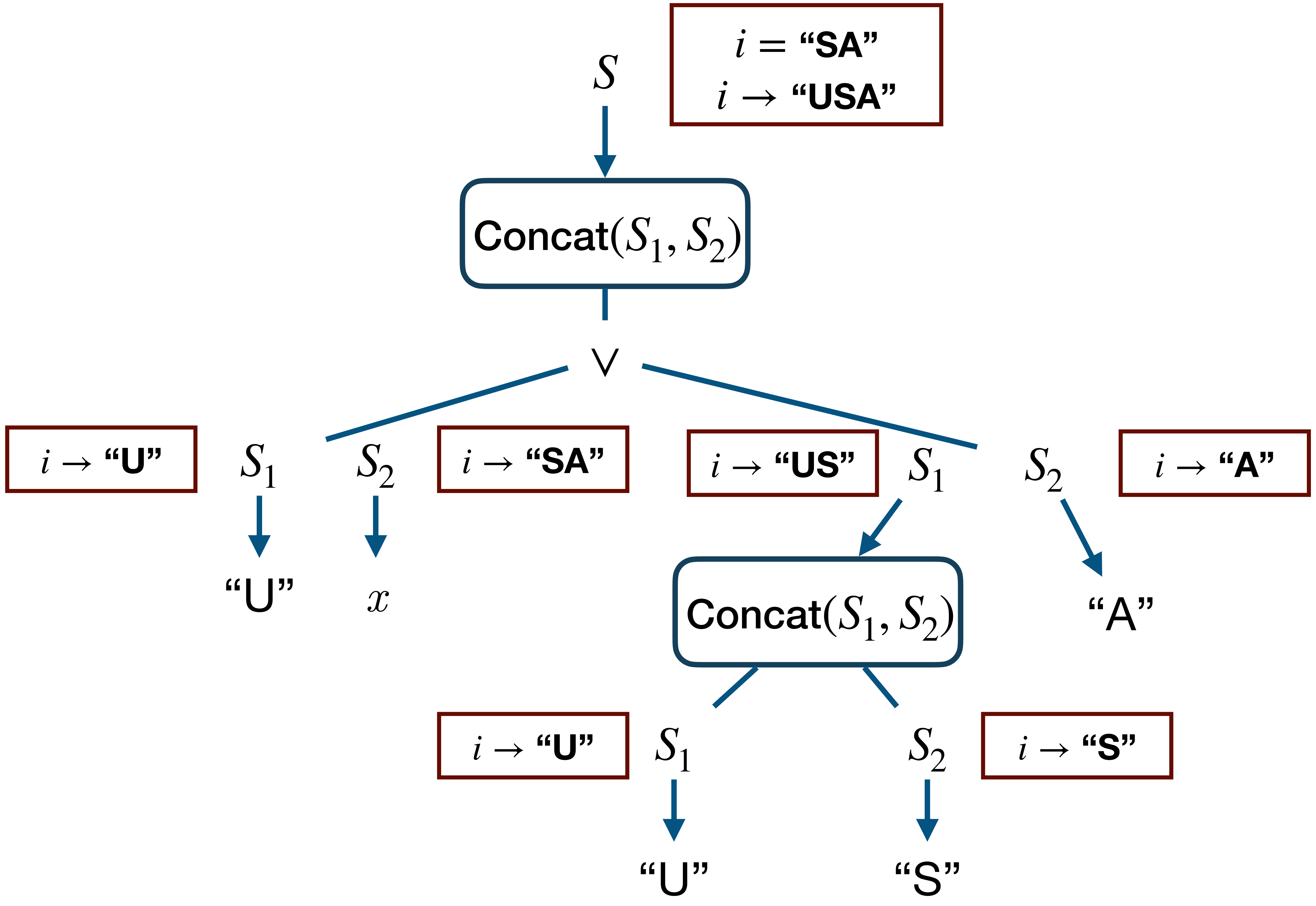


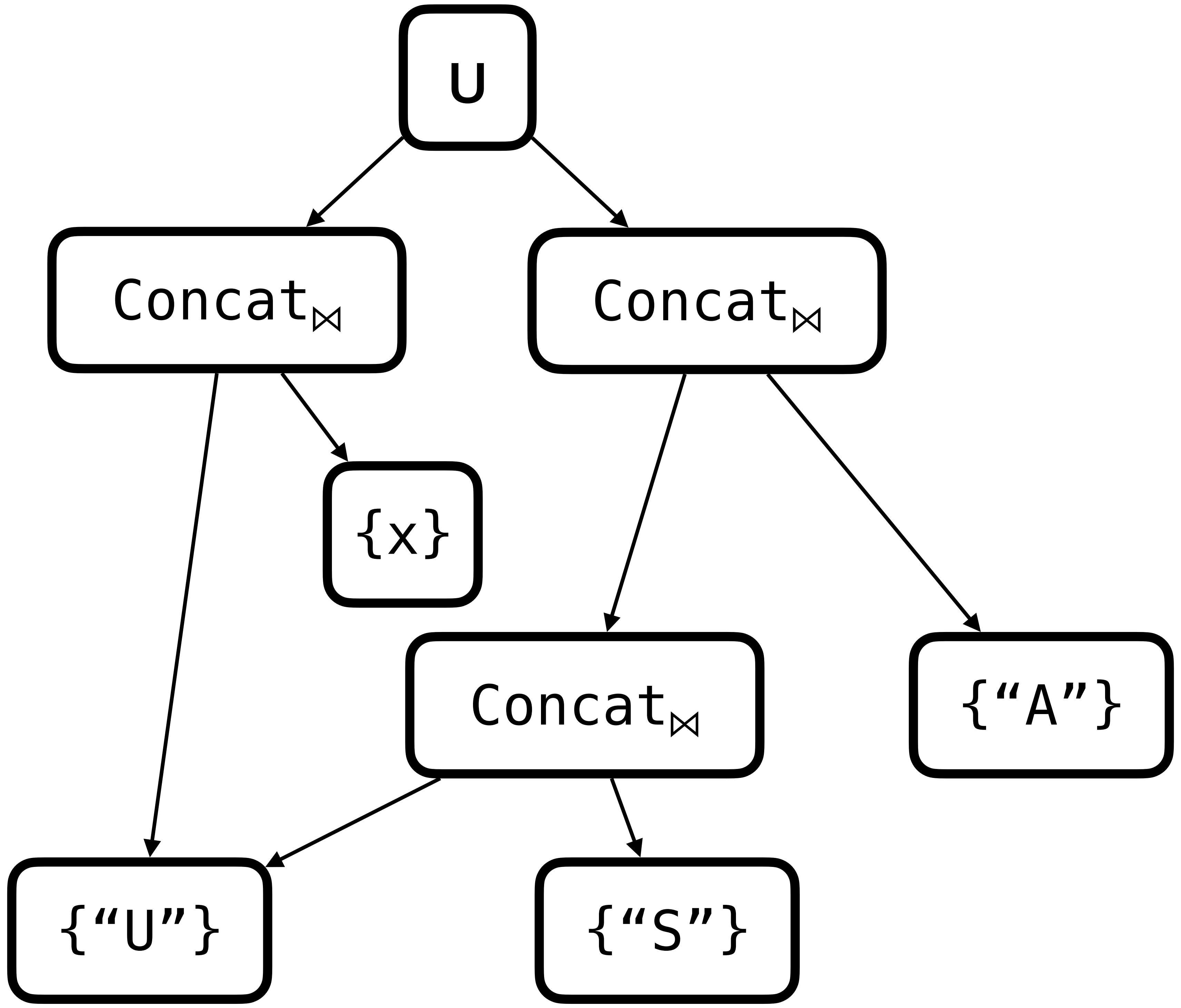
Example

- Grammar: $S \rightarrow ConstStr \mid x \mid \text{Concat}(S, S)$
- Specification: f (“SA”) = “USA” $\wedge f$ (“AE”) = “UAE”
- Inverse set:
  - Concat^{-1}(“USA”) = {(“U”, “SA”), (“US”, “A”)}
  - Concat^{-1}(“UAE”) = {(“U”, “AE”), (“UA”, “E”)}

Step 1-1: Learn

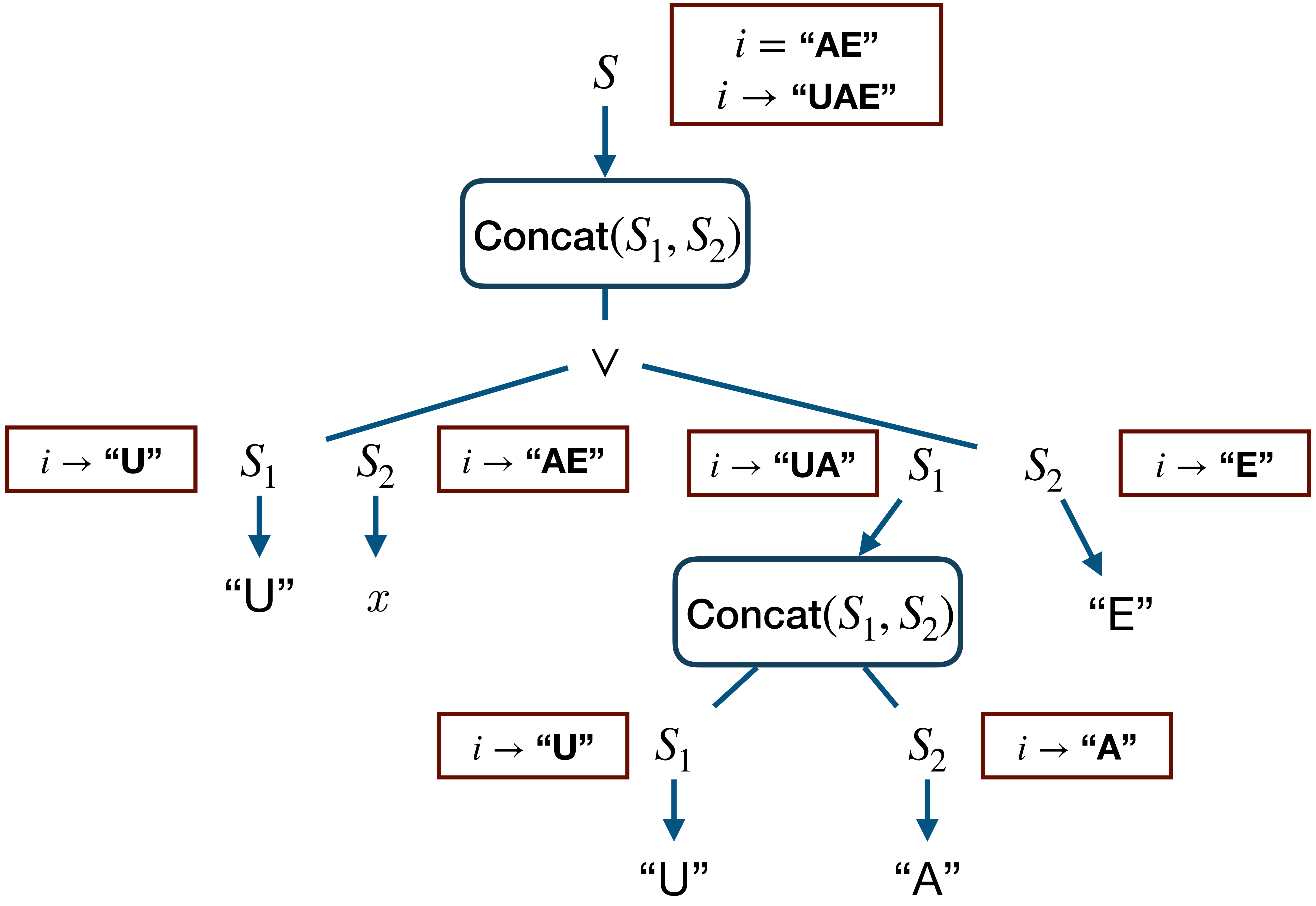
- Top-down propagation with one example

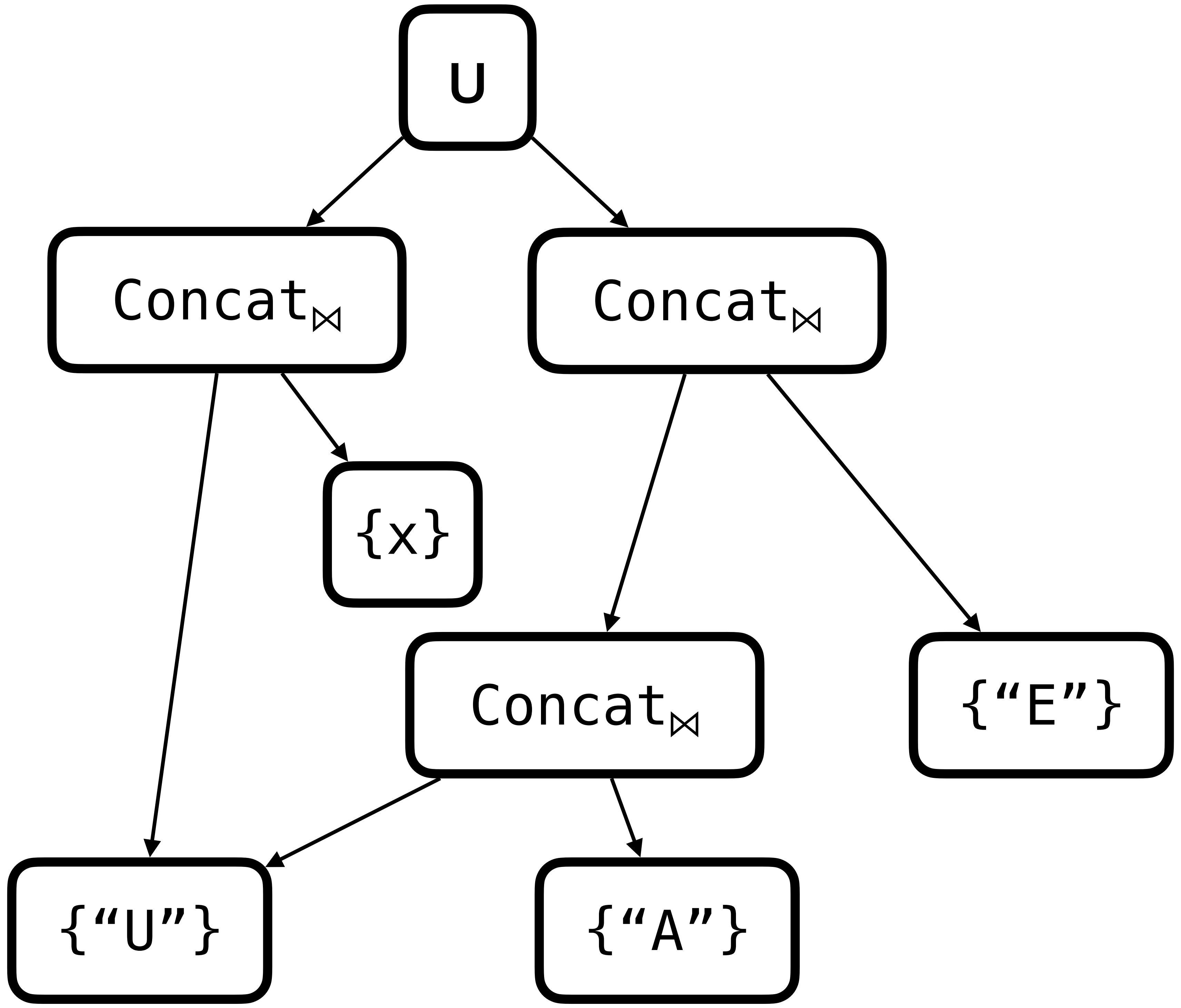




Step 1-2: Learn

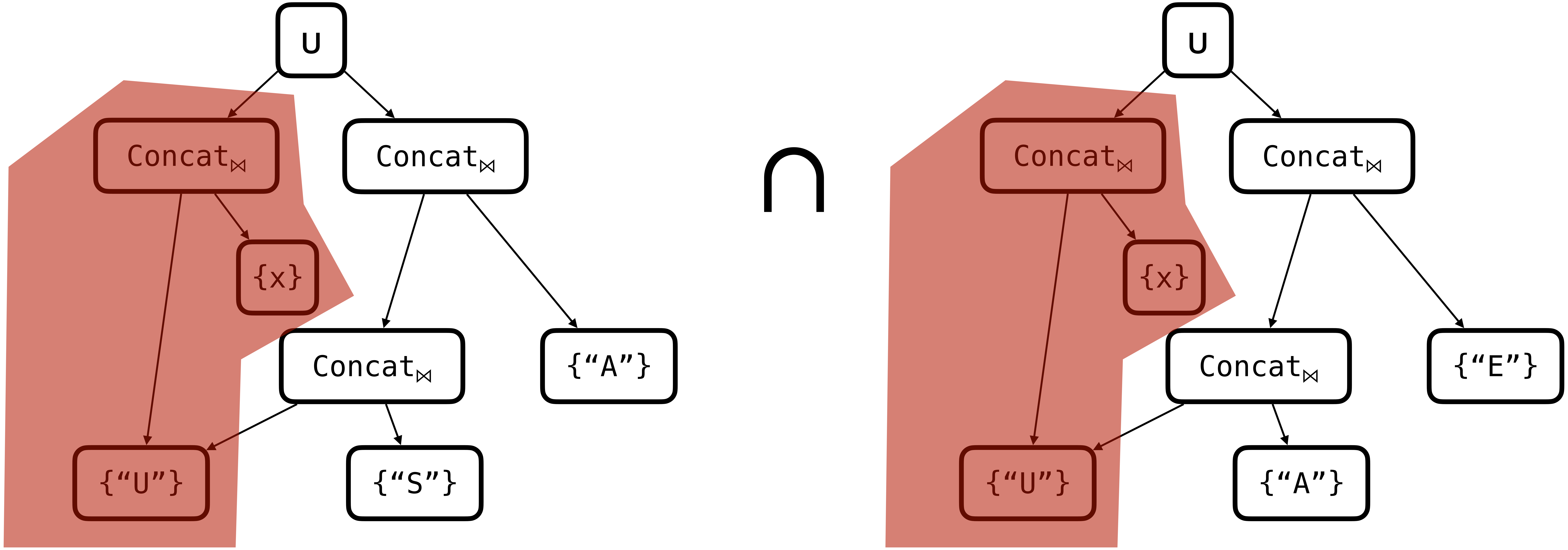
- Top-down propagation with next example





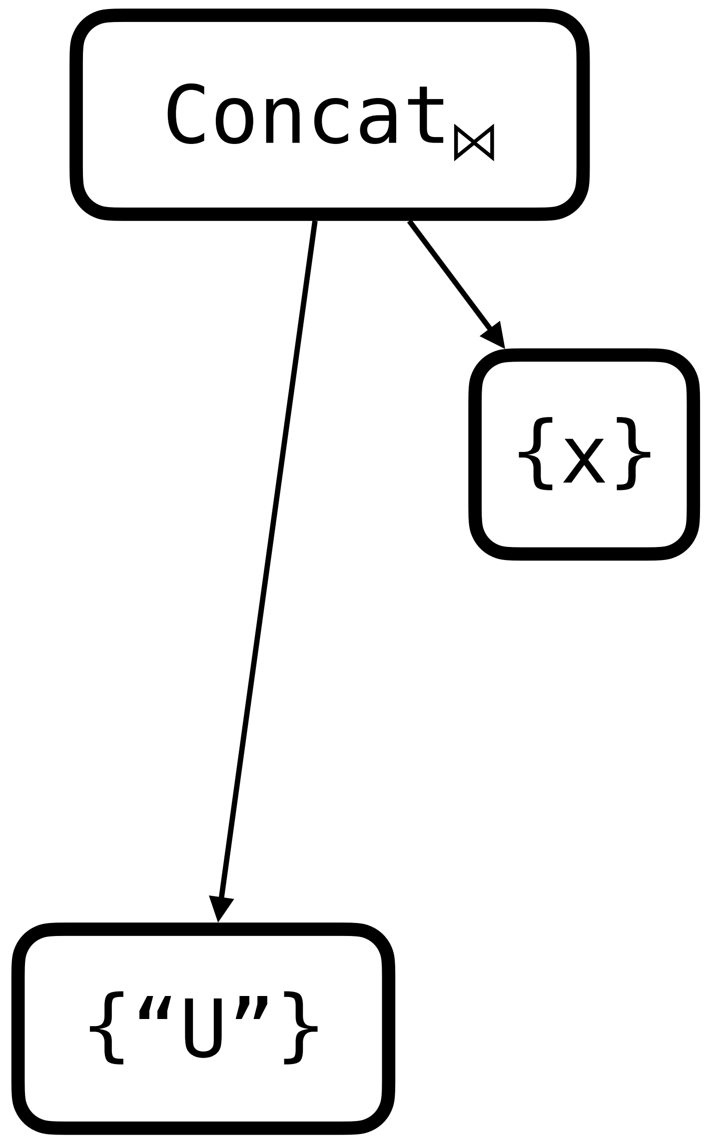
Step 2: Intersection

- Intersection of two version spaces



Step 3: Pick

- Pick a desired program



Concat(**“U”**, x)

Example

- Grammar:
  $S \rightarrow C \mid X \mid \text{Concat}(S, S) \mid \text{SubStr}(X, I, I) \mid \text{At}(X, I)$
  $I \rightarrow K \mid \text{IndexOf}(X, C, K) \mid \text{Length}(X)$
  $C \rightarrow$ “” | “ ”
  $X \rightarrow x$
  $K \rightarrow 0 \mid 1$

SubStr(s, i, n): longest substring of s of length at most n at i
E.g., SubStr(“KAIST”, 3, 5) = “ST”

- Specification: f(“Kihong Heo”) = “K Heo” $\wedge$ f(“Gildong Hong”) = “G Hong”
- Solution: $f(x) = \text{Concat}((\text{At}(x,0), \text{Substr}(x, \text{IndexOf}(x, \text{'' ''}, 0), \text{Length}(x)))$
- Inverse set: Concat^{-1}(“K Heo”) = {(“K”, “ Heo”), (“K ”, “Heo”), (“K H”, “eo”), …}
  At^{-1}(“K”) = {(x, 0)}
  SubStr^{-1}(“ Heo”) = {(x, 7, 4), (x, 7, 5), …, (x, 7, 10)}
  IndexOf^{-1}(7) = {(x, “ ”, 0)},
  $\text{Length}^{-1}(10)$ = {x}

(x, 7, n) where n > 10 never possible according to the grammar

Pros and Cons

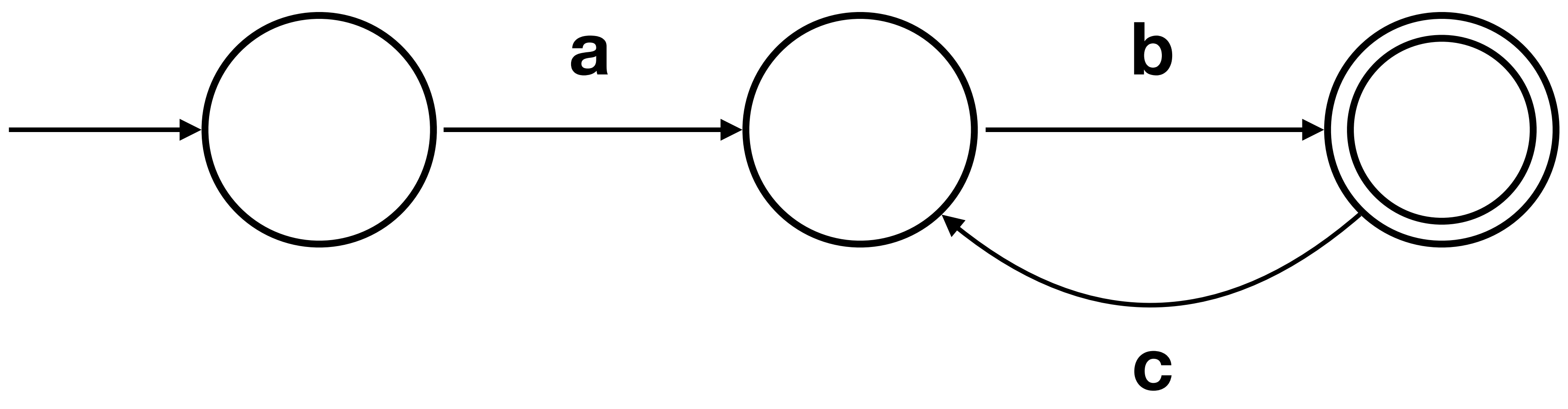
- Pros: efficient
  - Applications: Excel, VSCode, etc
  - See https://www.microsoft.com/en-us/research/group/prose/
- Cons: not always applicable
  - Efficiently computable inverse function
  - Finite inverse set

Representation-based Search

- Idea:
  - Build a data structure that concisely represents a set of programs
  - Extract solutions from that data structure
- Two well-known methods
  - Version space algebra (VSA)
  - Finite tree automata (FTA)

Automata

- Abstract models of machines
  - Computation: given an input, move through a series of states
  - Interest: the computation eventually halts at certain final states
- Many instances
  - Finite automata, push-down automata, …, Turing machine
- Example



{ab, abcb, abcbcb, …} : ab(cb)*

$S \rightarrow aA$
$A \rightarrow bB$
$B \rightarrow cA$
$B \rightarrow \epsilon$

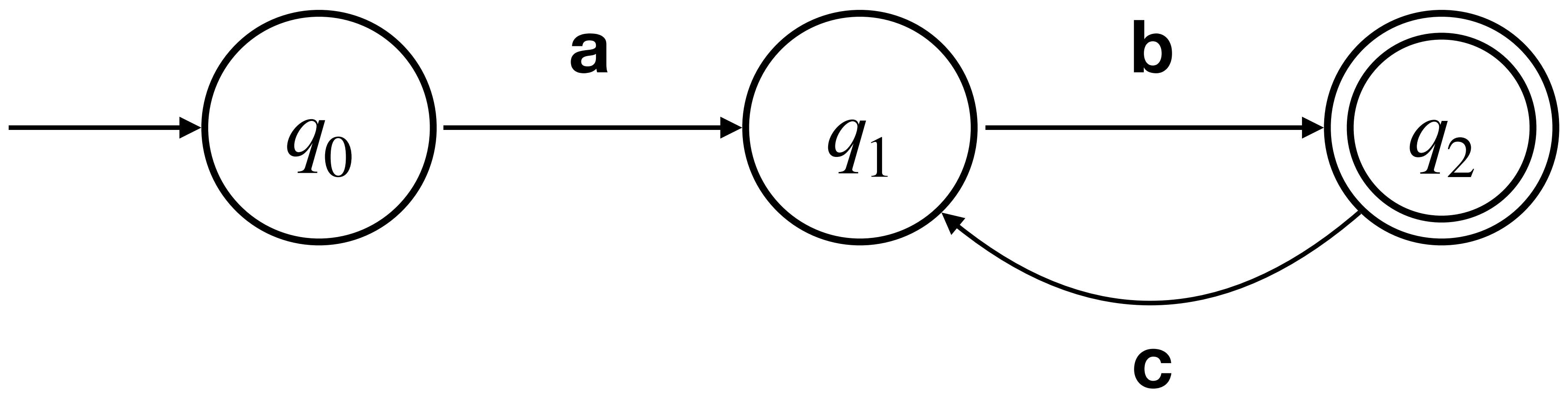
Example: Finite Automata

States **Initial state** **Final states**

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_f \rangle$$

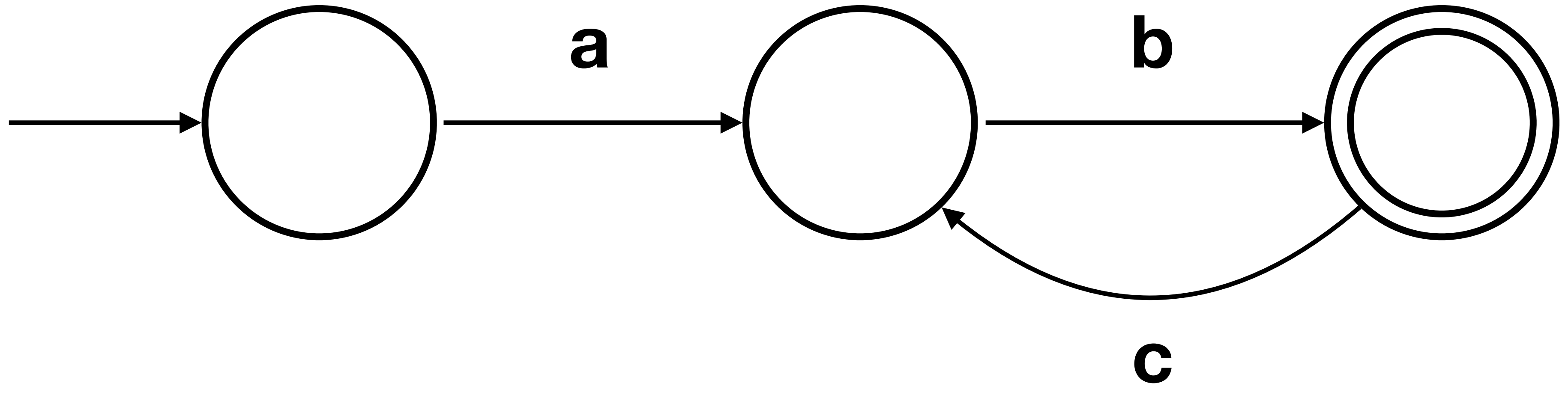
Alphabet **Transitions**

- $Q = \{q_0, q_1, q_2\}$ and $Q_f = \{q_2\}$
- $\Sigma = \{a, b, c\}$
- $\delta = \{(q_0, a, q_1), (q_1, b, q_2), (q_2, c, q_1)\}$



Why Automata in Synthesis?

- An automaton corresponds to a grammar
  - I.e., a set of input strings accepted by the automaton (or the grammar)
- A compact data structure for a set of programs
- Idea: bottom-up search via automata
  - Build the smallest automaton corresponding to a subset of the input grammar
  - Grow the automaton gradually according to the grammar



{ab, abcb, abcbcb, …} : abc*

$S \rightarrow aA$
$A \rightarrow bB$
$B \rightarrow cA$
$B \rightarrow \epsilon$

Example

Specification

Find a function $f(x)$ where $f(1) = 9$

Grammar

$$N \rightarrow \mathtt{id}(V) \mid N + T \mid N \times T$$
$$T \rightarrow 2 \mid 3$$
$$V \rightarrow x$$

Example

id(x) $*$ 3 $*$ 3

id(x) + 2 + 3 + 3

Finite Tree Automata

States **Initial state** **Final states**

$$\mathscr{A} = \langle Q, \Sigma, \delta, q_0, Q_f \rangle$$

Alphabet **Transitions**

- Example

Find a function $f(x)$ where $f(1) = 9$

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

$V \rightarrow x$

$Q = \{N, T, V\} \times \mathbb{N}$

$Q_f = \{\langle N,9 \rangle\}$

$\Sigma = \{\texttt{id}, + , \times \}$

$f(q_1, \ldots, q_n) \rightarrow q$

$\delta = \{\ \texttt{id}(\langle V,1 \rangle) \rightarrow \langle N,1 \rangle$

$+(\langle N, 1 \rangle, \langle T, 2 \rangle) \rightarrow \langle N, 3 \rangle$

$\times(\langle N, 1 \rangle, \langle T, 2 \rangle) \rightarrow \langle N, 2 \rangle$ **...** $\}$

Bottom-up Search with FTA

$$\delta = \{\ \mathtt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle$$
$$+(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 3\rangle$$
$$\times(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 2\rangle\ \ \ldots\ \}$$

Specification

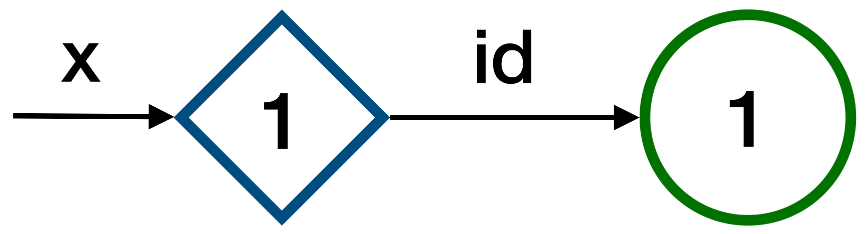
Find a function $f(x)$ where $f(1) = 9$

Grammar

$N \rightarrow \mathtt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

$V \rightarrow x$



$\mathtt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle$

Bottom-up Search with FTA

$$\delta = \{\ \texttt{id}(\langle V, 1\rangle) \rightarrow \langle N, 1\rangle$$
$$+(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 3\rangle$$
$$\times(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 2\rangle \quad \ldots\ \}$$

Specification

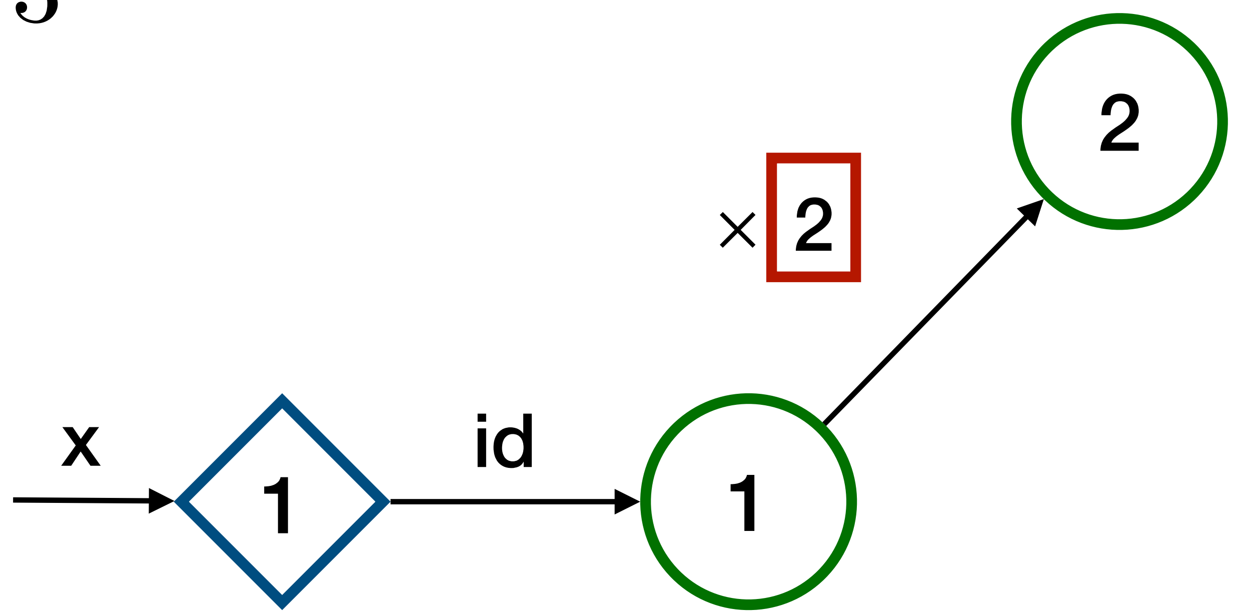
Find a function $f(x)$ where $f(1) = 9$

Grammar

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

$V \rightarrow x$



$$\times(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 2\rangle$$

Bottom-up Search with FTA

Specification

Find a function $f(x)$ where $f(1) = 9$

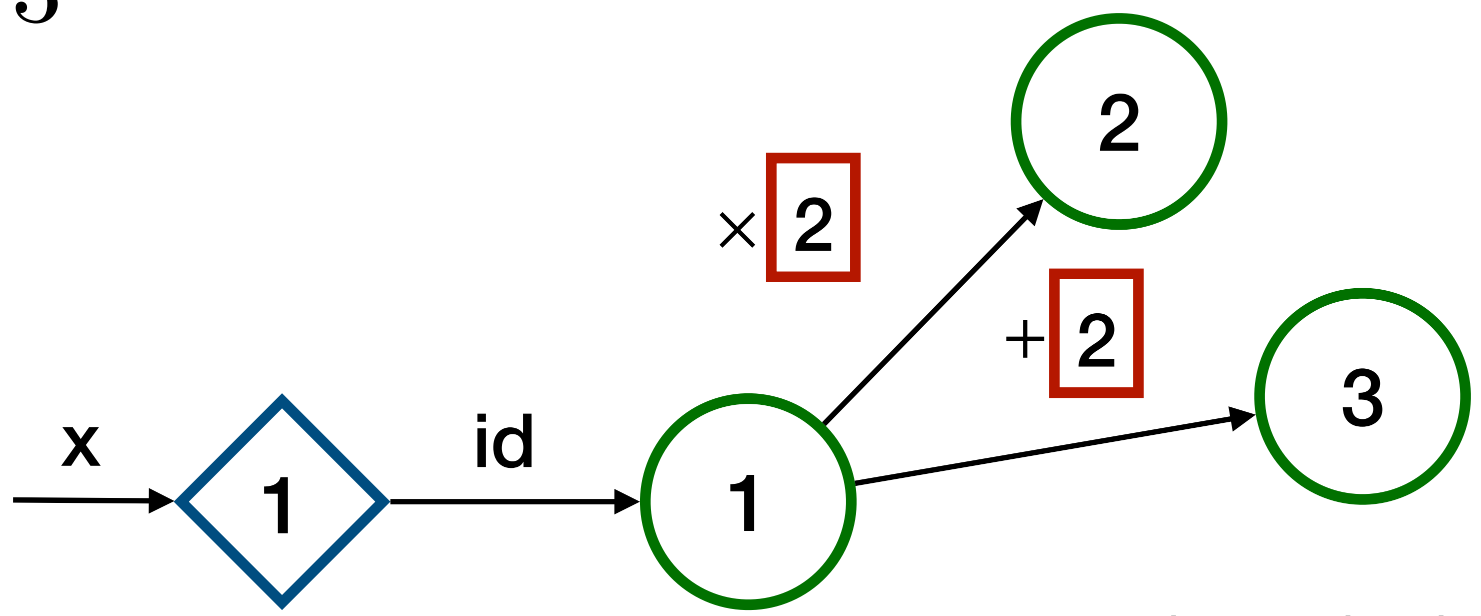
$$\delta = \{\ \mathtt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle$$
$$+(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 3\rangle$$
$$\times(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 2\rangle\ \ \ldots\ \}$$

Grammar

$N \rightarrow \mathtt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

$V \rightarrow x$



$+(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 3\rangle$

Bottom-up Search with FTA

Specification

Find a function $f(x)$ where $f(1) = 9$

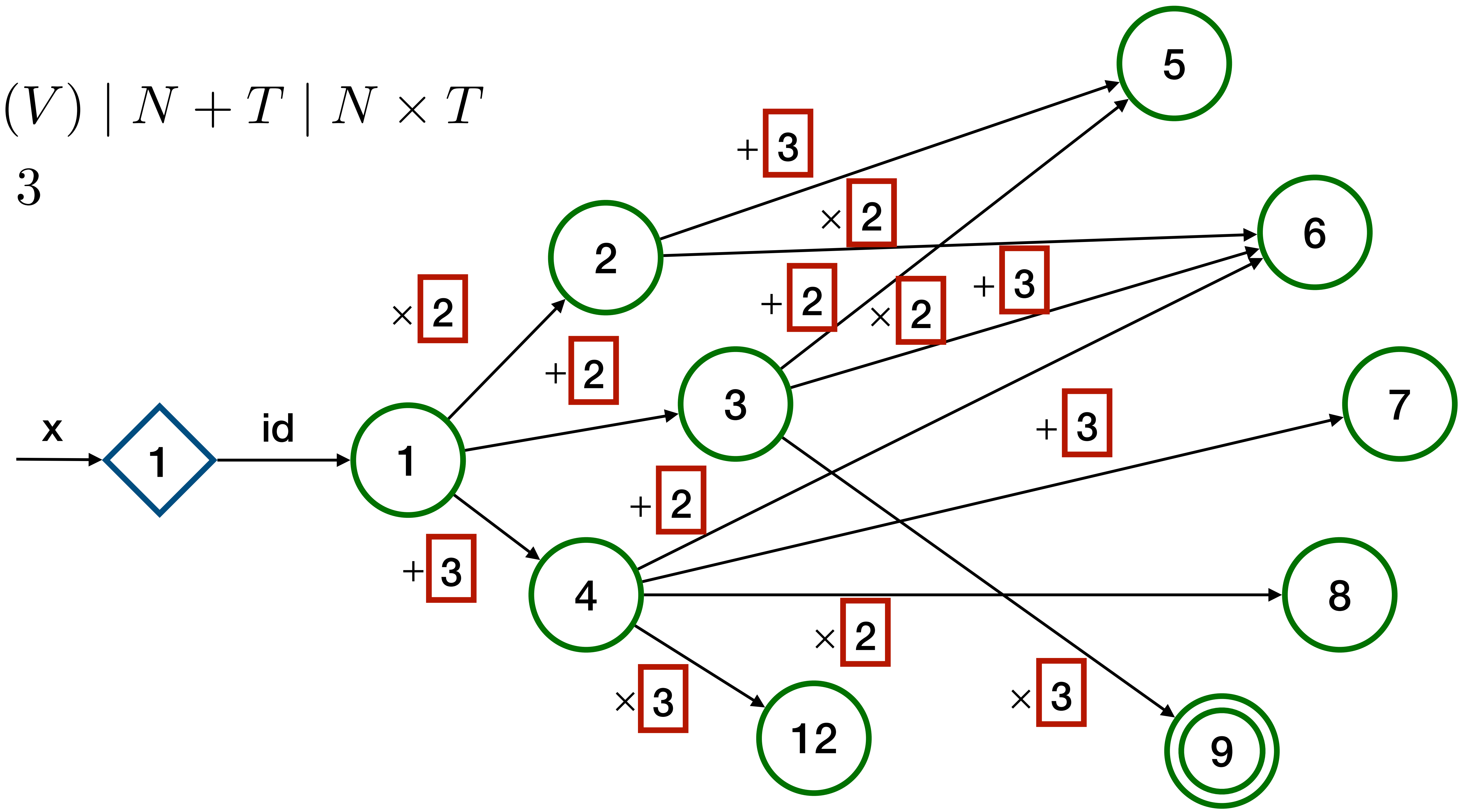
$$\delta = \{\ \texttt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle$$
$$+(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 3\rangle$$
$$\times(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \rightarrow \langle N,\ 2\rangle\ \ \ldots\ \}$$

Grammar

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

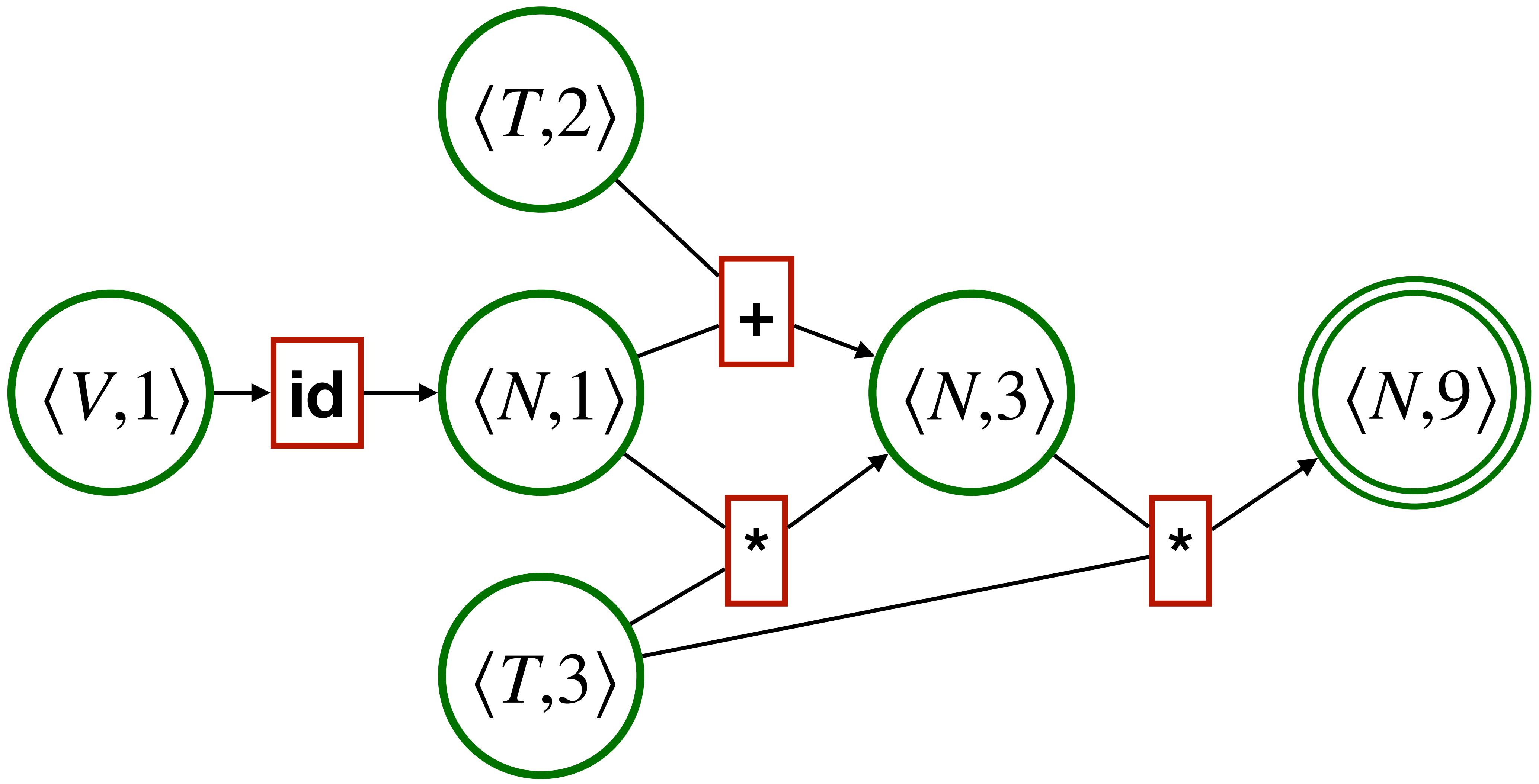
$V \rightarrow x$



FTA as Hypergraph

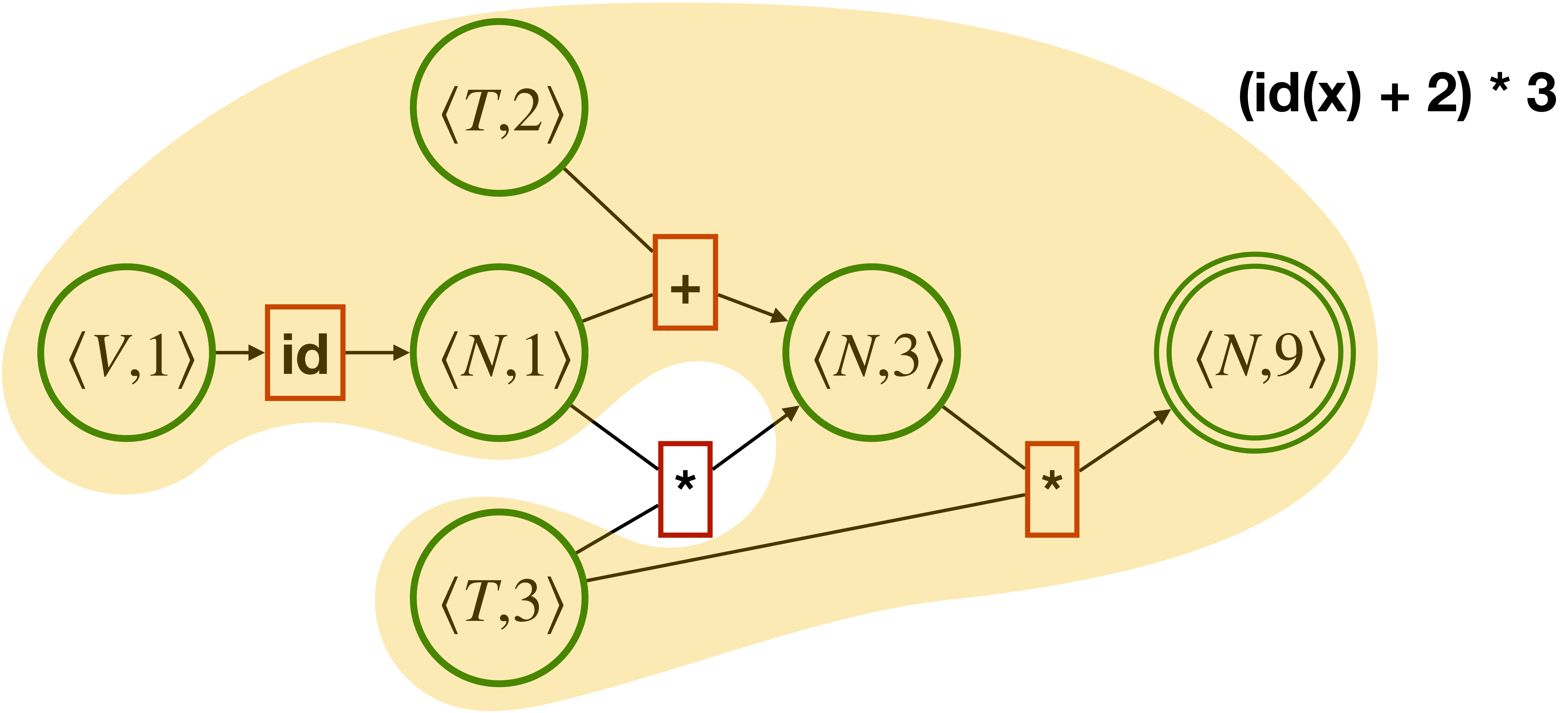
- Represent an FTA as a hypergraph (a generalization of graphs)
    - Nodes: FTA states
    - Edges: FTA transitions ($\wp(Node) \rightarrow Node$)

$$\delta = \{\ \mathtt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle$$
$$+(\langle N,1\rangle,\langle T,2\rangle) \rightarrow \langle N,3\rangle$$
$$\times(\langle N,1\rangle,\langle T,2\rangle) \rightarrow \langle N,2\rangle \ \ \ldots\ \}$$



FTA as Hypergraph

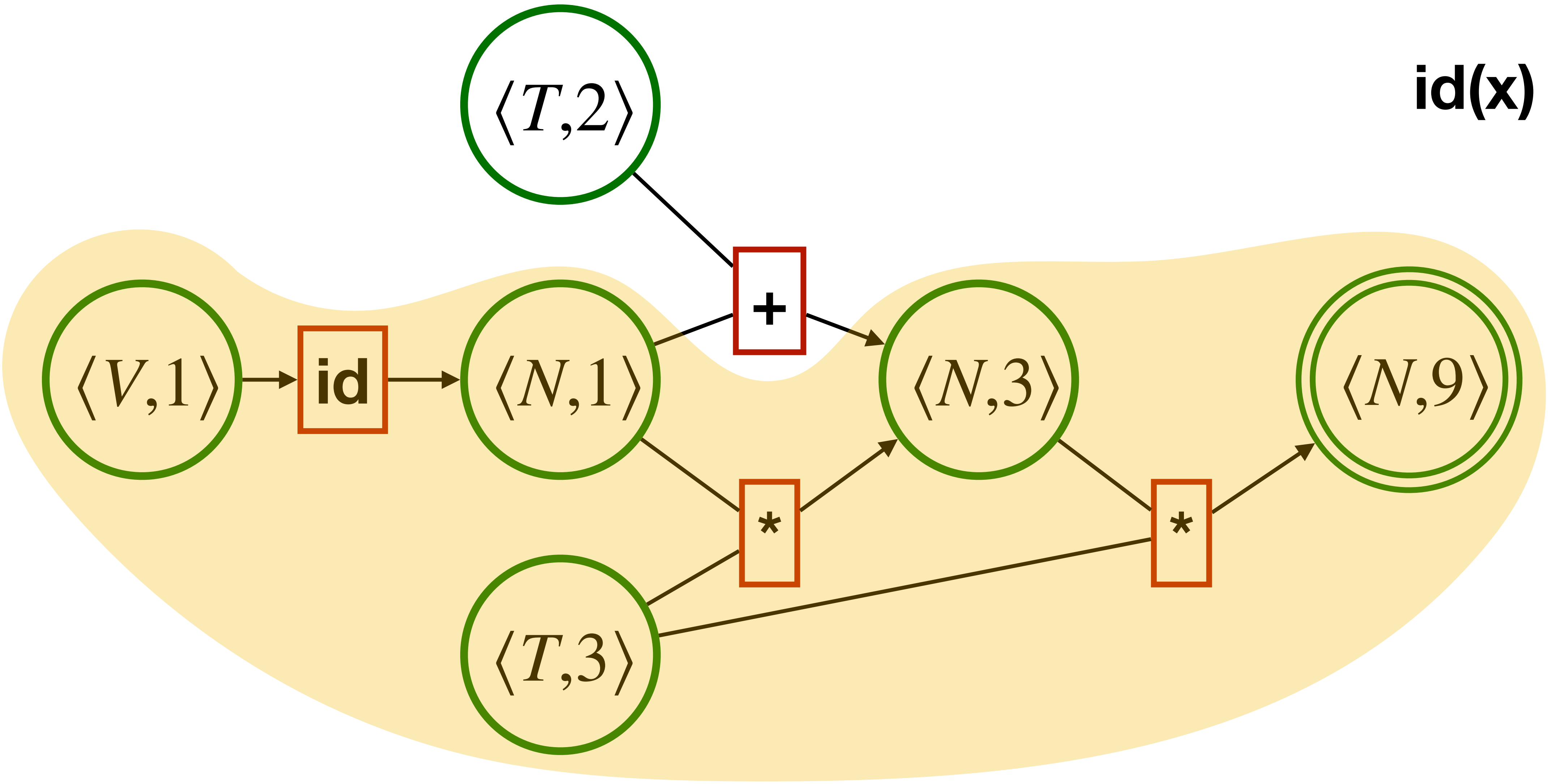
- Represent an FTA as a hypergraph (a generalization of graphs)
  - Nodes: FTA states
  - Edges: FTA transitions ($\wp(Node) \rightarrow Node$)



$$\delta = \{\ \texttt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle \\ +(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 3\rangle \\ \times(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 2\rangle \ \ \ldots\ \}$$

FTA as Hypergraph

- Represent an FTA as a hypergraph (a generalization of graphs)
  - Nodes: FTA states
  - Edges: FTA transitions ($\wp(Node) \rightarrow Node$)



id(x) * 3 * 3

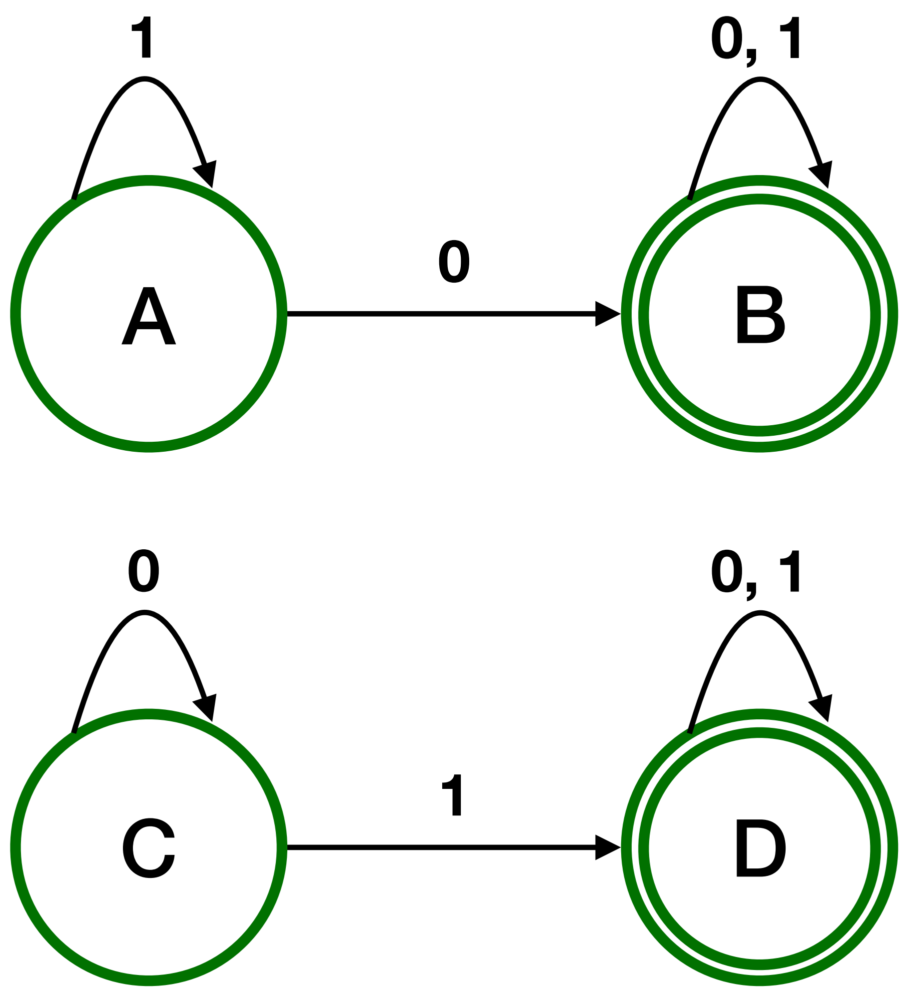
$$\delta = \{ \ \texttt{id}(\langle V,1\rangle) \rightarrow \langle N,1\rangle$$
$$+(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 3\rangle$$
$$\times(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 2\rangle \ \ \ldots \}$$

Other Practical Aspects

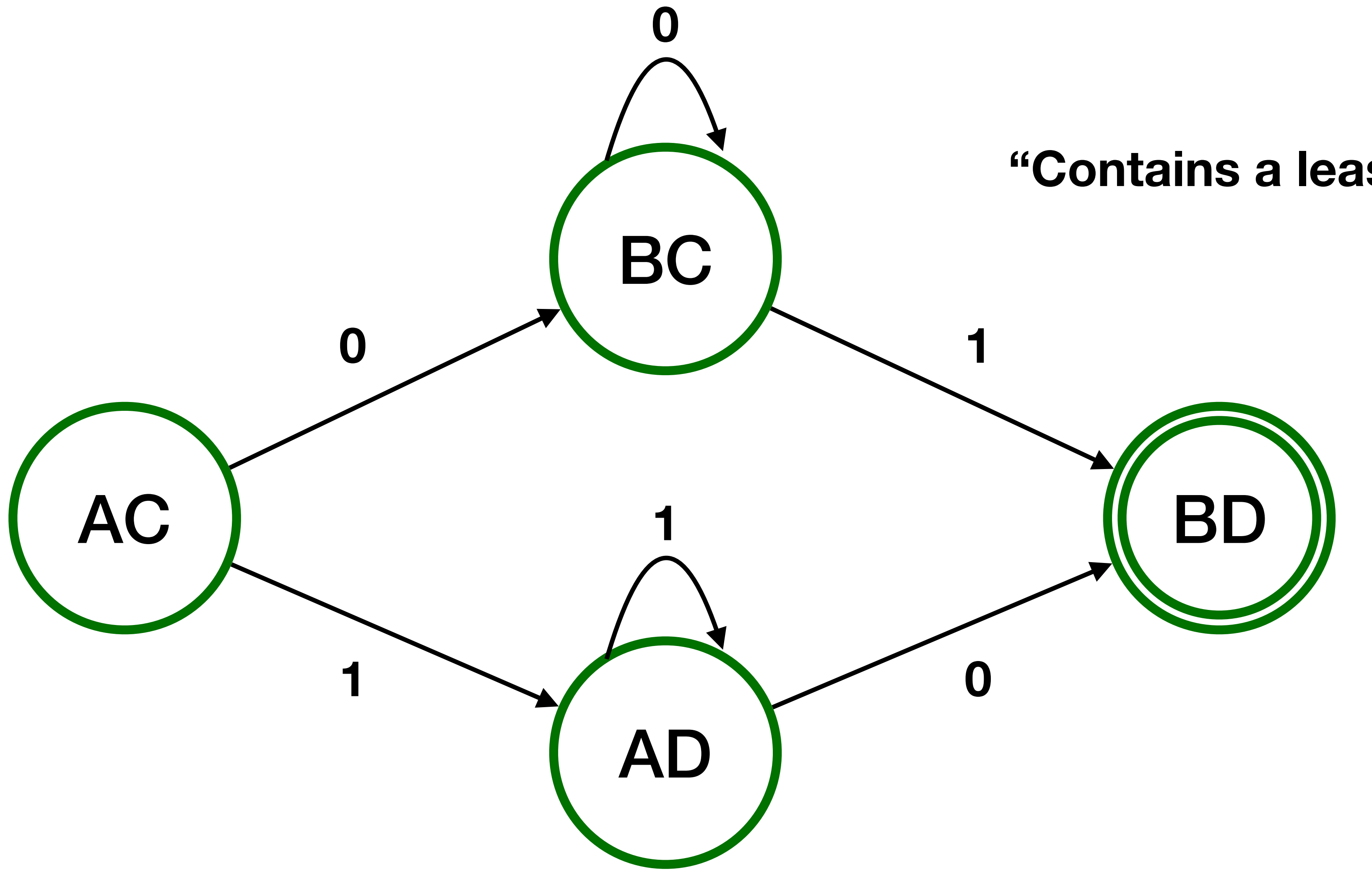
- Infinitely many states: usually limit the number of states (size of programs)
- Multiple examples: construct one automaton per example and compute their intersection
  - Use the standard method (more details in [CS322 Formal Languages and Automata])
  - Example

"Contains a least one 0"

"Contains a least one 1"



∩



"Contains a least one 0 and 1"

Summary

- Representation-based search
  - Search with space-efficient data structure
  - Represent multiple programs within a simple representation
- Combination with other search strategies
  - Version space algebra + top-down search (TDP)
  - Finite tree automata + bottom-up search